# A Program-Driven Parallel Machine Simulation Environment*

Chien-Chun Chou Po-Zung Chen Shyh-Nong Chen

Department of Computer Sciences and Information Engineering,
Tamkang University, Taipei, Taiwan, ROC

chou@cs.tku.edu.tw pozung@cs.tku.edu.tw csnong@cs.tku.edu.tw

## *Abstract*

*In recent years it has been very popular to employ the discrete-event simulation as a hardware architecture analytical tool to study the distributed-memory multicomputer and shared-memory multiprocessors. After the hardware architecture prototype has been completed, a complete and detailed machine simulation environment can be utilized to evaluate the architecture's efficiency under the real operating systems and application software. In this article, all the development, and implementation of a program-executable Transputer Network multicomputer as well as 80x86 series multiprocessors, and how they can be operated will be discussed. On another level, owing to the extreme complexity of the simulated computer systems, parallel discrete-event simulation has also been used to shorten the time of running the simulation. In practice, this simulator can solve problems through the network connection with many workstations. Some of the workstations may be in charge of computing, while some others can be responsible for the management of memory, thus making it simpler to establish a parallel machine simulation environment. In addition to providing an environment for programs to execute on it, such simulator will also calculate the time spent in running these programs, so as to evaluate the feasibility for these application programs to run on a hardware system.*

**Keywords :** discrete-event simulation, distributed-memory multicomputer, shared-memory multiprocessor, program-driven, Transputers Network

## 1. Introduction

With the rapid improvement in the development of computer hardware, both distributed-memory multicomputer and shared-memory multiprocessors have been very popular. In theory, the system which can collect the most nodes (processors) together is likely to have a larger memory bandwidth and better calculating ability of processor, and then to provide the best raw parallelism.

Because of the increasing number of nodes in multicomputer systems, however, many anti-efficiency problems have arisen in the interconnection network in the systems. And how to discover the way to produce the best interconnection network at the lowest price has always been an important moot point for research. The discussion includes what shape will be the best network architecture, which routing methods should be applied by the network routers [1] [2], whether it is proper to employ the wormhole [3] [4] switching techniques, whether it is necessary to connect nodes and their corresponding routers through multi-ports, and how the collective communication among nodes can be dealt with [5].

The problems arising oftenest in shared-memory multiprocessors, owing to the sharp increase in the number of processors, lie in how to make the most of bus, crossbar, or the connective architecture of multistage interconnection in order to effectively connect processors, caches, and shared-memory units and to control the multicache coherence of the units. The further challenge lies then in how to face the scalability problems [6] [7].

Many analytical models have been used to evaluate in advance the efficiency of the interconnection network architecture in multicomputers [8], the efficiency of the bus-protocol which protects the multicache coherence in shared-memory multiprocessors [9], and so forth. But, in response to the increasing complexity of these computer systems, it has been inevitable to employ some simplified hypotheses of analyzed systems in order toobtain the result of the analytical model analysis. In this way, some limits on a number of prerequisite conditions may have been set. During recent years it has been quite popular to employ discrete-event simulation instead as another analytical tool used for complex computer hardware

*This work was sponsored by grants from National Science Council, Taiwan, ROC (NSC 85-2213-E-032-007 and NSC 86-2213-E-032-002) and its partial results were published in [Chou97] [Chou98].

together with software. For example, the MultiSim (or MultiSim++) [10] of Michigan State University was adopted for research into the impacts that different interconnection network topological architectures had produced on the efficiency of multicomputer systems. On the other hand, the SimOS [11] of Stanford University is a more complete and detailed shared-memory multiprocessor simulator. Not only does it simulate the entire system, but also it allows actual software -- such as operating system [12] and distributed database application programs -- loaded on it and operated as if on a real hardware machine. Consequently, it will help evaluate, after the architecture prototype has been finished, the efficiency shown in the real workload of operating systems as well as application programs. Also, it will help develop, produce, and improve operating system software and application program software before hardware has been manufactured [11].

Likewise, on account of the increasing complexity of the simulated computer system and the fact that users always hope to acquire the evaluation result in the shortest simulation time, SimOS offered two kinds of different techniques, and it satisfies users' need by adopting them alternately. One of the techniques is a machine emulator which is faster but may be in distortion. In this case, SimOS imitates the result of the implementation of computer systems merely through dynamic binary translation rather than amply simulates the internal operations in hardware. And therefore it may not be possible to obtain all the evaluation data we want [13]. The other model uses machine simulation technique to simulate in detail the internal action in hardware and collect all the simulation results [11].

In this article we attempt to introduce the development, implementation, and prospect of the machine simulation environment of a multicomputer system and a multiprocessor system. At present, provided have been a Transputers Network multicomputer machine simulator that can execute INMOS T805 machine code [14] [15], a CPU that can execute Intel 80x86 machine code program [16], and multicache units. And they are accompanied with a single-bus to access shared memory. Meanwhile, we regard parallel discrete-event simulation as most important in shortening the simulation time [17] [18]. In implementation, yet, we should first connect a number of workstations through the Ethernet as a implementation platform for the simulator. At the same time, a distributed micro-kernel operating system that controls Transputers Network hardware resources is being developed. We will not only experiment with running this operating programs on the Transputers Network hardware that has virtually 17 nodes, but also we will load this micro-kernel into the Transputers Network hardware for simultaneous test and evaluation.

In the second section we will describe the the structure of the simulation environment, and in the third section we will describe in detail the implementation of Transputers Network simulators. In the fourth, we intend to introduce the implementation of 80x86 machine simulators. The fifth section is devoted to work afterwards and the future prospect. Finally, in the sixth section, we will add our bibliography as well as illustrations.

## 2. The structure of the simulation environment

In our machine simulation environment, the ultimate goal has been to evaluate, under the implementation of real operating systems and application software, the efficiency of different multicomputer and multiprocessor hardware and the software systems executed. This is by no means an easy task. As Figure 1 has shown, the entire machine simulation environment can be divided into three main levels. The upper level is the execution of programs generated by real workload (will include such complex software as operating system). The middle is the heart of the simulator, which part has to provide detailed simulators for each node of hardware model and to provide the simulator components of network interconnection among nodes. Up to now INMOS T805 node simulation, link simulation which owns 8 one-way synchronization point-to-point connections, and Intel 80x86 node simulation have been provided. Other components such as wormhole switching router and ALU-biasing network resource control will be established step by step. The lower level is the machine platform the whole simulator will implement on. Because we shorten the total simulation time by way of parallel discrete-event simulation, in implementation we establish the present implementation platform first by connecting many workstations through Ethernet. But in the future it will be taken into consideration to employ other more advanced multicomputers and multiprocessors as our operation platforms.

## 3. The implementation of the transputers network multicomputer machine simulation environment

Throughout the implementation of the machine simulator, it is C language that develops those programs executed on the hardware platform, like Sun Sparc5/SunOS and IBM RS6000/AIX workstations, and the basis of the simulation are the on-chip interconnection network architecture units and CPU in the INMOS T805 Transputer node. Put simply, a Transputers Network is in fact a hardware realization of Communicating Sequential Processes [19].

Simulators are produced by means of discrete-event simulation, which can provide program debugging in programs, simulation time calculation, and efficiency analization. Basically, a simulator will choose the first event each time from the pending-event set to simulate the operation and calculate the simulation time, and if necessary, it will insert a new generated event into the set. It will repeat the process until a predetermined simulation time is expired or there is no more event in the set. A pending-event set is one used to store all the data architecture of the events that are going to happen but

haven't happened and haven't been simulated. The stored events are ordered according to the recorded time they will happen. Moreover, the data architecture includes not only sorts of events and recorded time, but also a WorkspacePtr field to record the memory workspace address at which the process of the event happened (which will be detailed in section 3.1).

The major events of simulators are FETCH, DECODE, SCHEDULE, DESCHEDULE, RESCHEDULE, WAITTIMER, WAKEUP, and EXTERNALEVENT. Among them, FETCH event can have the four instructions of the consecutive four bytes pointed by Instruction pointer register (Iptr) in memory being copied to the prefetch register. DECODE event will execute the instructions (saved in advance in prefetch register) pointed by Iptr, and, when executing the fourth instruction in prefetch register or doing branch jump, it will add a FETCH event to the pending-event set. SCHEDULE event stands for the time slice of a low-priority process has expired. This process will no more have the control over CPUs and will be moved to the end of the active queue set to which the process belongs, and at the same time, the first process of the pending-event set will be given the control of CPUs. DESCHEDULE event indicates that process will release the control over CPU and will be moved out to be a process in waiting state. RESCHEDULE event will move the process in waiting state back to the end of the pending-event set. WAITTIMER event will move the process to the end of the time queue and wait for the arrival of a certain time. WAKEUP event denotes the arrival of a certain time and the process will be moved back to the end of the set. EXTERNALEVENT represents an external interruption event has been generated [20].

In the following sections a more detailed introduction will be given to major parts of a simulator, like the memory and workspace in nodes, pending-event set, and interconnection network architecture simulation.

### 3.1 The simulation of memory and workspace

A Transputer node contains a 4K-byte T805 on-chip memory, adding with its unique memory in the node up to a continuous 4G bytes memory space. The entire memory space starts from the 32-bit negative minimum $80000000_{16}$ and end with the positive maximum $7FFFFFFF_{16}$. The time for accessing these two kinds of memories is different, and hardware DisableIntRAM pin is necessary in choosing between the two. When any other node's private memory is needed, it can be accessed only through the network hardware channel.

In a simulator we represent 4K on-chip memory by Memory[] array. On the other hand, when nodes are communicating with the help of hardware channel, all this is done through serial network linker. The hardware units of this hardware architecture and the memory in the nodes have the input/output capability like Direct Memory Access, and thus the data can be read from or write to memory without intervention by the CPU. With different UNIX processes we will simulate the hardware action of the CPU and serial linker in the nodes respectively. Yet the memory simulation array Memory[] is designed with the shared memory mechanism of UNIX, which after being controlled by semaphore, can be used by CPUs and serial linker at the same time.

It is the workspace in Transputer node that can store frequently used variables in process context. And the methods of accessing data in the workspace are all operated by workspace pointer register (Wptr) as a basis then adding the offset (one byte per unit) to it. Further, once the workspace is established in the on-chip memory, there will be a difference of three times to once in the time it may take to access data comparatively in the unique memory of the nodes. Besides, some significant information concerning process status is stored in the workspace. Such information lies in the zone below the place Wptr pointed at. For instance, Iptr field will point to the address of the next instruction to be executed; Link field may point at the workspace of the next process in the active queue; Pointer field may point to the destination or resource address of the data communication (used in communication through channel). Also, TLink field points to the workspace of the next process in the timer queue. Time field then stores the time that process has spent waiting.

### 3.2 The simulation of the active queue

Transputer node provides process with two levels of executive choices: high priority and low priority. So there are two active queues, which are respectively used to store the execution order for high priority processes and low priority processes. For each active queue there are a front-end pointer register and a back-end pointer register, i.e. Fptr0 and Bptr0 registers, point respectively to the workspace address of the first as well as the last process of the high priority queue. (Fptr1 and Bptr1 registers are on the low priority active queue in the same way). Besides, as stated in the preceding section, every process's workspace has a Link field which can point to the next process's workspace, thus forming the structure of the active queue.

Each time when process is put into the active queue, the CPU will first keep the value of the Iptr register of this process in the Iptr field of its workspace and then put the workspace of this process into the active queue. In Transputer nodes the high priority process can preempt the execution of the low priority process at any time, and only when waiting for channel I/O will the high priority process release the execution power. Except when being preempted or waiting for channel I/O, low priority process has to release its execution power as time expired. All these related scheduling procedures are completed in hardware implementation through microcode.

### 3.3 The simulation of interconnection network architecture

In the Transputers Network, the communication between two processes is made possible through channel which posses the point-to-point, one-way, buffer-less, and synchronized characteristics. Also, channel provides real-time communication ability of input and output, required in parallel programs. The so-called channel is in fact a word in memory. Quite different from traditional communication ports, its only function is to store the workspace address of those processes that are ready for communication through this channel, rather than store the information to be transfer. Therefore, when process is communicating through channel, it is unnecessary to write data into channel. And besides, at any given time, there can be only one process waiting for communication in a channel. Whether the two processes waiting for communication are in the same node affects the way they will communicate. When the two processes are in the same node, this channel is called the soft channel or internal channel. If the two processes are in the different node, however, this channel is called the hard channel or external channel.

To compare soft and hard channels, both have basically the same logical behavior but different efficiency. In an actual hardware architecture, because the two processes communicating through soft channel are situated at the same Transputer node, they achieve parallel processing through time-sharing. The moving of memory during communication is done through CPU. While the two processes communicating with hard channel are in the different nodes, which can easily execute in parallel. And the moving of memory during communication is done through the serial linker among nodes using DMA hardware, which can work with CPU at same time without occupying the CPU time. In Transputers Network simulator, the simulation of hardware channel communication is made possible by UNIX fork() function call to produce new UNIX processes and to realize the parallel machine simulation of CPU and serial link.

### 3.4 Calculation of the simulation time

The simulation time can basically be classified into the CPU instruction execution time, the memory accessing time, the process scheduling time of microcoded scheduler, the interruption time, and the linker communication time.

The calculation of the simulation time spent on CPU is done through accumulating instruction execution time, which can provide important information for related research into the efficiency on analyzing and execution programs hereafter. When several processes are concurrently time-sharing processing in CPU, the simulation time means all the time needed for the system simulation after the simulator operations, rather than the time for only one process. In addition, we have to accumulate the memory accessing time, for example, – if a on-chip memory accessing needs one clock cycle, then it will be three clock cycles for external memory accessing.

As for the calculation of the interruption time, according to the manual [15], if an interruption happens, it may require 53 to 58 clock cycles doing context switch. Under the situation of FPU, it may require about 78 clock cycles doing context switch as an interruption happens. For the communication time, the calculation has been based on the input and output instructions given by the manual [15]. The communication between "in" and "out" instructions takes 2w+19 clock cycles, in which w represents how many bytes needed for the communication messages. And the communication time spent in "outbyte" and "outword" instructions is 23 clock cycles.

### 3.5 The parallel processing architecture and the implementation of simulators

Here we are going to discuss how to realize the architecture of, when distributed on different platforms, the node simulations of Transputers Network simulator. And also, how to simulate serial linker communication through IPC mechanism of UNIX.

#### 3.5.1 The network resource description file

The network resource description file is a text file describing effective network hardware. We choose a language rule offered by [21] for users to describe the network architecture they desired. The content of a network resource description file of has to be in following language format:

```
"network" <name>
  "{"
    ( "processor" <name> "{" <connections> ";" <opts>
    "}" )+
  "}"
```

<name> after the key word "processor" is the identification for each node in network. And <connections> is the connection method of four serial linkers. The optional <opts> is used to explain the size of the memory and the program files this node is about to run.

#### 3.5.2 The set-up and operation of the parallel processing architecture

Basically, Transputers Network simulators generates, according to the network resource description file, all the nodes to be simulated on the network by means of UNIX fork() function call. And when all the nodes are running on the same computer, they can choose either with the UNIX pipe mechanism or the UNIX domain stream socket mechanism to simulate the whole channel. When all the simulated nodes are distributed to different

computers to be executed, channels are simulated by UNIX Internet stream socket mechanism (note: it has to be determined in advance at compile-time, that is to use single-workstation or multi-workstation platform, or use UNIX IPC mechanism). In another aspect, when simulators are being run, each node will initialize the state variables of memory module and register module (As Figure 2 has shown) and, according to the network resource description file, will design the network connecting architecture to be simulated. Eventually, the CPU simulation in every node of the simulator will load, according to the network resource description file and then starts the simulation.

Presently, we have not evaluated efficiency or improved the execution speed simulated in single-workstation or multi-workstation platforms (we have only testified the execution result of the simulation), but all this is going to be part of the further plan.

## 4. The implementation of 80x86 multiprocessor machine simulation environment

During the implementation, a simulator can be divided into two parts: CPU and memory unit. Of them memory unit can also be classified into cache-memory unit and shared-memory unit. This research offers program code of C language, tested on Sun Sparc5 which runs SunOS 4.1.3.

Concerning CPU, the 80x86 series designed by Intel has been simulated. Because this research is focused on the computing efficiency of application program running in shared memory, we use 8088 simulator, designed by Tanenbaum, as the basis [16], and we have modified the program code and seen the "records of used memory," which accessed the main memory, as the input of our "CPU/cache-memory/shared-memory" simulator. These records all based on the physical addresses. In the aspect of accessing cache memory, we adopted prefetch algorithm. And the alternatives as well as mapping methods have been dealt with set-associative mapping, accompanied with the Least Recent Used (LRU) algorithm.

We are going to adopt the program-driven and sequential discrete-event simulation as our strategy to simulate multiprocessor machine. The target of this simulator is a machine equipped with more than two CPUs. Each CPU has its own cache-memory unit, which communicate with a shared-memory unit via a single-bus system. In this respect we will solve any data coherence problems between all the cache memory and shared memory. As Figure 3 shows the relationship among every module. The simulator first reads a architecture description file from a architecture-description module and, founds a systemic architecture according to the parameter obtained. Then, the loading module reads the program code of an application program and puts it into the architecture. Finally, the kernel module begins the simulation, according to the systemic architecture and the program code of application programs, and it will produce statistic information together with the final result of recording the memory.

The research below will offer a more detailed introduction to major parts of the simulator, like CPUs, memory units, data coherenc, Spinlock simulation, and so forth.

### 4.1 The simulation of CPUs

In the simulation programs of CPUs, originally designed by Tanenbaum, there is a simple main-memory architecture. Tanenbaum declared a 1MB one-dimension array to be the execution space for the system. This memory-unit architecture is too simple to exhibit layers architecture between cache memory and main memory. In our implementation, we designed a memory-accessing function as interface through which CPUs and memory units can communicate, consequently eliciting memory units completely from CPUs.

About the simulation of multiprocessor in question, we attained the goal principally with the fork() system call supported by UNIX. As soon as the process is created, a socket channel to memory units is built up. As long as the communication channel between CPUs and memory units has been established, the variable, sockfd[], becomes the identification code, by way of which memory units can clearly recognize which CPU sent the request received. Hereby, memory units can access the corresponding cache-memory units.

### 4.2 The simulation of memory units

Memory units include cache-memory units and shared-memory units. In the aspect of accessing cache memory, we adopted prefetch algorithm. And the alternative as well as mapping methods have been dealt with set-associative mapping, accompanied with the Least Recent Used (LRU) algorithm. Eventually, the updating of the information about main memory as well as cache memory has been done through write-through method.

As long as memory units have been elicited they will be simulated by a UNIX process. When the CPU simulating process is calling the memory-accessing function, the communication with memory units will go on through socket provided by UNIX. As cache-memory units receive data from CPUs in the simulator, they will start searching and accessing data. Inasmuch as main memory plays a very passive part throughout the system, only when cache memory needs to access data in main memory will it do it through the function call.

The whole cache-memory architecture is constructed and surmounted with linked list and dynamic memory allocation. Such design is perfectly suitable for adjustment of cache-memory design by means of parameters. Further, the chief data architecture of main

memory units is to declare a 1MB one-dimension array (provided that the system has 1MB memory space) and that is responsible for offering data to cache-memory units.

### 4.3 The operational architecture of simulator and the data coherence

We take the architecture with three CPUs as a example, there are three CPU places with one token on each. And they share a token in the bus free place. When CPUs need to access cache memory, the token on CPU place will arrive the cache access places. If now the communication with other cache memory or shared main memory via bus is not needed, then tokens will just go back to CPU places. Otherwise, the tokens would have reached get bus places and checked whether there was any token on bus free place. In case when the token reaches get bus place and find no token there, it means the control of bus has been taken over by other cache-memory units, and they have to wait until there is a token on the bus free places (that is to say, other cache-memory units have released the control of bus), and then the fight for the control will begin. Once the control gained, the tokens in get bus places will arrive master places and begin to use bus.

### 4.4 The maintenance of the data coherence

When different CPUs need to access the same address in shared memory, we will positively need a set of rules to assure the data coherence between every cache-memory unit and shared memory units. In our simulator, the maintenance of the data coherence is complemented by a mixed strategy of Firefly protocol and IEEE Futurebus+ standards. With this strategy, the updating of information about cache-memory units has been write-through. While the upgrading of data between cache-memory units and shared main memory units is write-back.

### 4.5 System parameters

Our simulator can adjust the system architecture parameters, such as the number of CPUs, characteristics of cache memory, and certain related time parameters. The parameters related to the cache-memory includes the number of bits of divided cache-memory set, the number of bits in every cache-memory page (page stands for reservable pages meanwhile in each collection), and the number of bits of the size of each page in cache memory. Time-related parameters are: the basic CPU processing time, the data transfer rate between CPU and cache-memory units, the time for cache memory to read and write, the data transfer rate in cache memory and between cache memory and shared memory, the time for shared memory to read and write, and the time for shared memory to deal with addresses sent from cache memory.

### 4.6 The realization of multiprocessor system with sequential simulation method

At present we simulate multiprocessor system by using sequential strategy. We illustrate with four processes running in the simulator simultaneously (Take three-CPU systems for example. Three processes take charge of simulating three CPUs, one of simulates memory units.). Now we gather together all the control over shared-memory units, cache-memory units, and the data coherence inside one process to execute.

In the course of the simulation, every CPU has a variable (a or b) to record simulation time, and in memory units there is a variable used to record system simulation time. When memory units receive two events sent from CPU, they will choose an event of less local simulation time to simulate (suppose $a < b$) and push the system simulation time into this shorter local simulation time (that is, the event sent from CPU, whose local time is a, will be simulated first, and the system simulation time will be pushed to a.). In choosing an event to be simulated, however, anything can be started only after one condition has been satisfied: memory units have to collect each event sent from CPU will it be able to simulate one of them, and remove it from the list of events. In other words, if a CPU stops sending events to memory units, then the events sent from another CPU can by no means be simulated. In order to avoid such things, every CPU must send at least an empty event every other few minutes to maintain the implementation of the system.

There are five kinds of events in the system: Read Cache, Write Cache, Read Miss Transaction, Write Miss Transaction, and Write Hit Transaction. A Read Cache event happens when CPU wants to read data in memory units. A Write Cache event happens when CPU wants to update data in memory units. When a Read Cache event is simulated, if it is a Read Hit, then data will be read and sent back to CPU. If it is a Read Miss event here, then a Read Miss Transaction event will happen . When a Write Cache event is simulated, if it is a Write Hit, then a Write Hit Transaction event will happen. If it is a Write Miss here, then a Write Miss Transaction event will happen.

### 4.7 The simulation of spinlock

When an application program is running in a parallel processing environment, mutual exclusion is the first problem we need to solve. Our system provides the Spinlock function to solve this problem. First, we establish a test-and-set hardware simulation in shared memory, send the lock variable address to shared memory, read the value of lock variable, and set the value of lock variable to be 1. All the actions above can be finished within a clock cycle. And we also add three instructions -- test-and-set lock variable, load lock variable, and unlock lock variable – to CPU. Test-and-set lock variable is to get the value of lock variable into AL register, meanwhile setting the lock variable to be 1. Load lock variable is to get the value of lock variable into the AL register. Unlock lock variable is to set the lock variable to be 0.

## 5. The conclusion and the future prospect

Multicomputer machine simulation environment is not a dream any longer. An prototype of Transputers Network simulator, which can directly run INMOS T805 machine code, has been created now. And on this platform a distributed operating system that can control Transputers Network is being developed (as shown in Figure 4). Nevertheless, in order to create an intact multicomputer machine simulation environment, used to evaluate the operational efficiency of multicomputer hardware architecture fitted into software, there is still a long way to go. Here we merely present some fruits of the great work – it leaves still a lot to be desired. For instance, (1) much more interconnection network simulation components (such as wormhole switching router, and ALU-biasing-styled network resource control) are required. (2) Evaluational data have to be clearly defined and displayed. (3) Improvement as well as evaluation of the efficiency has to be made on the simulated execution speed of multi-workstation platform. But we strongly believe the multicomputer machine simulation under real workload will be able to provide from-up-to-down a complete and real evaluation environment, for original cross-section evaluation, to see this environment from. Comparatively, the complexity has been increased, but it may be the time now to overcome them.

As for the use of simulators, the present user-friendly interface has to be improved. When the simulator is at work, the system architecture relies still very much upon manual power. We hope to offer an integrated environment, given only some relevant information and then we can have a systematically well-designed environment. Up to now our machines have progressed only to an extent of sequential simulation. Thus we hope to properly cut each module of the simulator into parallel operational modules, in order to increase the execution efficiency of the simulator. Now we have only simulated a multicomputer and multiprocessor hardware architecture. In fact we still lack certain control over application programs run in the parallel processing environment. In the future research and development, we hope to strengthen the control mentioned here, so that we can evaluate more conveniently the execution efficiency of application programs. Ultimately, we hope in the near future we will be able to create a sound parallel environment on the net.

## 6. Bibliography and illustrations


[1] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," JACM, vol. 41, pp. 874-902, Sept. 1994.

[2] A. Chien, Kim, Horst, Krause, Sonnier, and Watson "Network resource control in the Tandem ServerNet router," submitted for publication, CSAG, University of Illinois at Urbana-Champaign, Mar. 1996.

[3] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," IEEE Computer, vol. 26, pp. 62-76, Feb. 1993.

[4] L. M. Ni, Y. Gui, and S. Q. Morre, "Performance evaluation of switch-based wormhole networks," Proc. Of the 1995 International Conference on Parallel Processing, vol. 1, Aug. 1995.

[5] Y. Tsai and P. K. McKinley, "An extended dominating node approach to collective communication in all-port wormhole-routed mesh networks," Technical Report, MSU-CPS-94-55, Michigan State University, Oct. 1994.

[6] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," Computer, pp. 12-24, June 1990.

[7] M. Heinrich, M. Rosenblum, J. Hennessy, etc., "The performance impact of flexibility in the Stanford FLASH multiprocessor," Proc. Of the 6th International Conf. On Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI), San Jose, CA, Oct. 1994.

[8] D. A. Reed and R. M. Fujimoto, "Multicomputer networks message-based parallel processing," The MIT Press, Cambridge, Massachusetts, London, England, 1987.

[9] M. Ajmone-Marsan, G. Balbo, and G. Conte, "Performance models of multiprocessor systems, " The MIT Press, Cambridge, Massachusetts, London, England, 1986.

[10] P. K. McKinley and C. Trefftz, "MultiSim: A simulation tool for the study of large-scale multiprocessors," Proc. Of the 1993 International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Networks (MASCOTS), pp. 57-62, San Diego, California, Jan. 1993.

[11] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Fast and accurate multiprocessor simulation: the SimOS approach," in IEEE Parallel and Distributed Technology, vol. 3, #4, Fall 1995.

[12] J. Chapin, M. Rosenblum, S. Devine, T. Lahiri, D. Teodosiu, and A. Gupta, "Hive: Fault containment for shared-memory multiprocessors," The 15th Symposium on Operating Systems Principles, Dec. 1995.

[13] E. Witchel and M. Rosenblum, "Embra: faster flexible machine simulation," The Proc. Of SIGMETRICS '96, pp. 68-79, May 1996.

[14] "The T805 transputer instruction set manual," 1st ed., INMOS Limited, 1991.

[15] "Transputer data book," 2nd ed., INMOS Limited, 1990.

[16] From ftp://netbsd.csie.nctu.edu.tw/pub/Minix/simulator

[17] J. Misra, "Distributed discrete-event simulation," ACM Computing Survey, Vol. 18, No. 1, pp. 39-65, Mar. 1986.

[18] Chien-Chun Chou, "Parallel simulation and its performance evaluation," Technical Report 93-02 (Ph.D. Thesis), Department of Computer Science, the University of Iowa, Iowa City, USA, Feb. 1993.

[19] C. A. R. Hoare, "Communicating sequential processes," CACM, vol. 21, pp. 666-677, Aug. 1978.

[20] Shyh-Nong Chen, "Transputers Network User Guide," Department of Computer Sciences and Information Engineering, Tamkang University, Taipei, Taiwan, ROC (in Chinese)

[21] "The Helios parallel operating system," Perihelion Software Ltd., 1991.

[22] Chien-Chun Chou and Shyh-Nong Chen, "A Program-Driven Multicomputer Machine Simulation Environment," Proceedings of 1997 Workshop on Distributed System Technologies & Applications. pp. 416-422 (in Chinese)

[23] Chien-Chun Chou, Po-Zung Chen and Shyh-Nong Chen, "A Program-Driven Multiprocessor Machine Simulation Environment," Proceedings of 1998 Workshop on Distributed System Technologies & Applications. pp. 368-375 (in Chinese)

[24] W. S. Colin, "The transputer," Proc. 12th Int'l Sym. of Computer Arch., IEEE, pp. 292-300, 1989.
[25] Institute of Electronics and Electronics Engineers, inc. "IEEE Standard for Futurebus+ -- Logical Protocol Protocol Specification," IEEE Std896.1, 1994

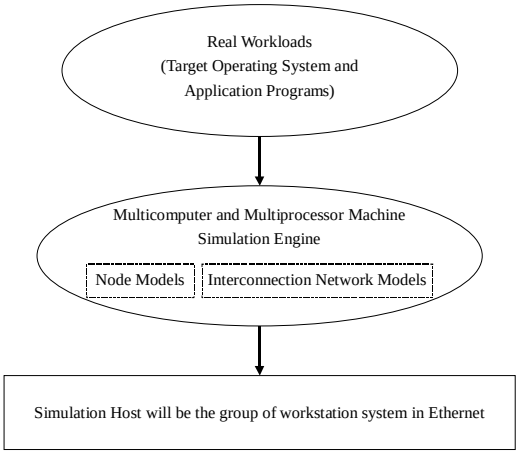


**Figure 1. A program-driven parallel machine simulation environment**

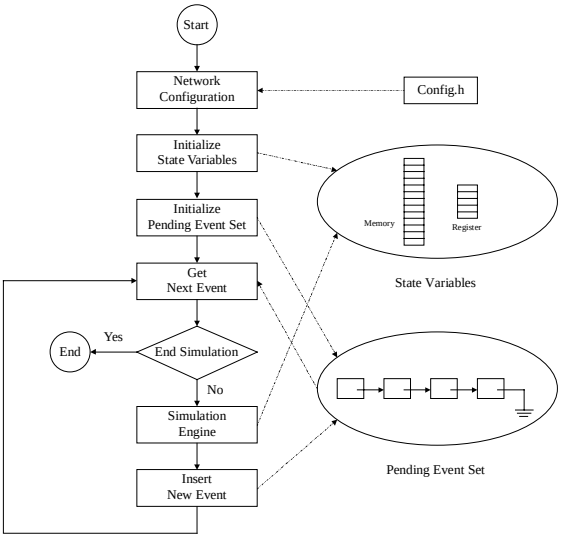


**Figure 2. Simulator Architecture Configuration**

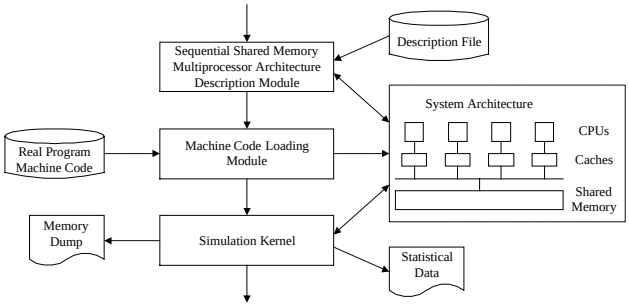


**Figure 3. Simulator Architecture Configuration**

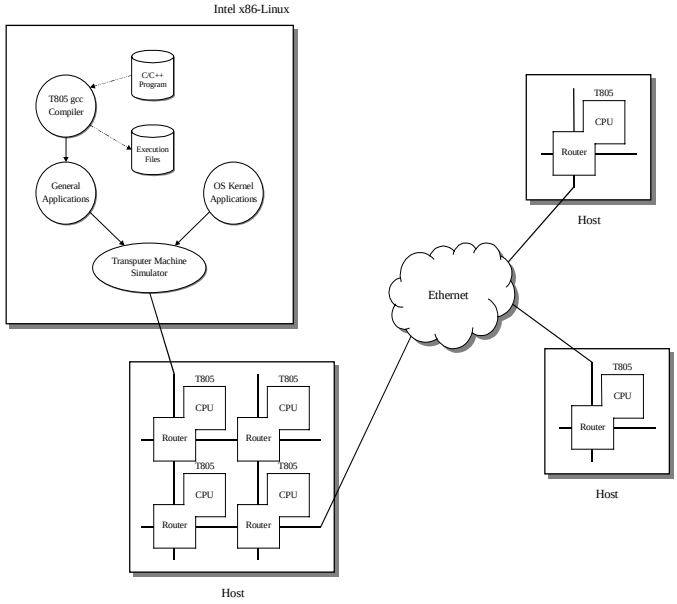


**Figure 4. Multicomputer Machine Simulation Environment**